Teaching Multiagent Systems: Past and Future

José M Vidal[1] Paul Buhler[2] Hrishikesh Goradia[1]

[1]Department of Computer Science and Engineering
University of South Carolina

[2]Department of Computer Science
College of Charleston

Teaching Multiagent Systems Workshop, 19 July 2004

Past Classes

- *Introduction to Multiagent Systems* graduate class.
- Taught six times between 1999–2003.
- 10–20 students each time.
- Used Weiss and Wooldridge textbooks.
- No prerequisites.
- Used RoboCup, Jade, FIPA-OS, and NetLogo as teaching tools.

Approach

- Multiagent research is divided into
  - **Theory and algorithms**: game theory, auctions, utility theory, distributed algorithms, logic.
  - **Software and hardware agents**: agent systems, ontologies, communications.

Approach

- Multiagent research is divided into
  - **Theory and algorithms**: game theory, auctions, utility theory, distributed algorithms, logic.
  - **Software and hardware agents**: agent systems, ontologies, communications.

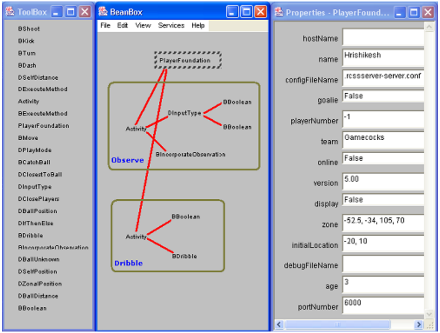
Approach: Let students build systems so they can see the algorithms in action and understand how local changes affect the emergent behavior of the system.

Using RoboCup

- Used RoboCup since second class.
- Students form teams of one to three students. Compete in tournament.
- Early lesson: need better basic agent.

Using RoboCup

- Used RoboCup since second class.
- Students form teams of one to three students. Compete in tournament.
- Early lesson: need better basic agent.
- Developed Biter and SoccerBeans.



hostName	
name	Hrishikesh
configFileName	.rcssserver-server.conf
goalie	False
playerNumber	-1
team	Gamecocks
online	False
version	5.00
display	False
zone	-52.5, -34, 105, 70
initialLocation	-20, 10
debugFileName	
age	3
portNumber	6000

  - Biter contains many basic behaviors (dribbling, passing, catching) and subsumption and BDI architecture support.
  - SoccerBeans turns these into Beans and allows the use of Sun's Bean Development Kit.

Lessons Learned

- RoboCup usage has had many benefits:
  - It is an easy problem to learn.
  - Students are very motivated to win and try different techniques.
  - Strategy is more important than raw performance (all teams play each other).
  - First-hand experience with nonintuitive emergent behaviors.

Lessons Learned

- RoboCup usage has had many benefits:
    - It is an easy problem to learn.
    - Students are very motivated to win and try different techniques.
    - Strategy is more important than raw performance (all teams play each other).
    - First-hand experience with nonintuitive emergent behaviors.
- But, it has some drawbacks:
    - Techniques developed for domain are unlikely to transfer to other domains.
    - Very few of the standard multiagent algorithms are applicable.
    - No selfish agents.

Lessons Learned

- RoboCup usage has had many benefits:
  - It is an easy problem to learn.
  - Students are very motivated to win and try different techniques.
  - Strategy is more important than raw performance (all teams play each other).
  - First-hand experience with nonintuitive emergent behaviors.
- But, it has some drawbacks:
  - Techniques developed for domain are unlikely to transfer to other domains.
  - Very few of the standard multiagent algorithms are applicable.
  - No selfish agents.
- Biter is essential but SoccerBeans was unsatisfactory due to problems with BDK.

NetLogo Background

- NetLogo is a programming language/environment used for modeling complex systems.
- It is a descendant of StarLogo which is a parallel version of Logo.
- Logo is a variant of Lisp designed to teach children basics of programming.

NetLogo Background

- NetLogo is a programming language/environment used for modeling complex systems.
- It is a descendant of StarLogo which is a parallel version of Logo.
- Logo is a variant of Lisp designed to teach children basics of programming.
- StarLogo was designed to teach children the **distributed mindset**.
  - We are born with a tendency to explain all phenomena, including emergent, by alluding to a central controller.
  - For example, kids think the Queen tells the ants what to do.

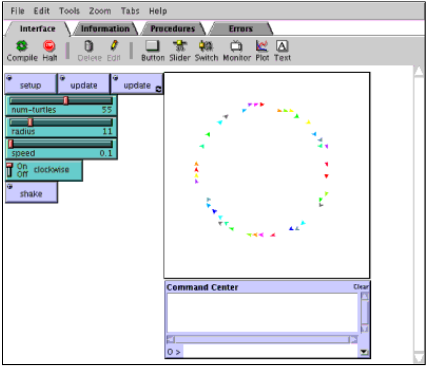
NetLogo Background

- NetLogo is a programming language/environment used for modeling complex systems.
- It is a descendant of StarLogo which is a parallel version of Logo.
- Logo is a variant of Lisp designed to teach children basics of programming.
- StarLogo was designed to teach children the **distributed mindset**.
  - We are born with a tendency to explain all phenomena, including emergent, by alluding to a central controller.
  - For example, kids think the Queen tells the ants what to do.
- NetLogo is written in Java and includes sophisticated primitives.

```
to setup
  ca
  create-n-turtles num-turtles
end

to move
  locals [cx cy]
  set cx mean values-from turtles [xcor]
  set cy mean values-from turtles [ycor]
  set heading towardsxy cx cy
  if (distancexy cx cy < radius) [
    set heading heading + 180]
  if (abs distancexy cx cy - radius > 1)[
    fd speed / 1.414]
  set heading towardsxy cx cy
  ifelse (clockwise) [
    set heading heading - 90]
  [
    set heading heading + 90]
  fd speed / 1.414
end

to update
  no-display
  while [count turtles > num-turtles][
    ask random-one-of turtles [die]]
  ask turtles [move]
  display
end
```



num-turtles 55

radius 11

speed 0.1

On Off clockwise

```
to create-n-turtles [n]
  create-custom-turtles n [
    fd random 20
    shake]
end

to shake
  set heading heading + (random 10) - 5
  set xcor xcor + random 10 - 5
  set ycor ycor + random 10 - 5
end
```

Other NetLogo Programs

1. Adopt algorithm for graph coloring and N-queens problem.
2. Asynchronous backtracking for N-queens.
3. Mailmen problem.
4. Tileworld problem.
5. Asynchronous weak commitment for N-queens.
6. Path-finding using pheromones.
7. Distributed recommender system simulation.
8. Reciprocity in package delivery.
9. The coordination game.
10. Congregating.

`http://jmvidal.cse.sc.edu/netlogomas/`

NetLogo Class Use

- One day introduction/demo of NetLogo and its history and purpose.
- Five or six two week long assignments using NetLogo.
- Implement known algorithm or solve open problem using techniques from class.

Lessons Learned

- NetLogo benefits:
  - Easy to learn.
  - Very short develop-test cycle.
  - Easy graphics, easy GUI development, lots of playing!
- Minor problems:
  - Hard to specify problem description in code.
  - Lack of object model created some confusion.
  - Students unfamiliar with list operators (map, reduce).

FIPA Agents

- We have used both JADE and FIPA-OS.
- Assignments consists of groups of 1–3 students building an application such as a distributed meeting scheduler.
- Each agent would need to cooperate with other in order to maximize its own utility.
- The students had to develop their own communication protocols which the agents had to obey.

Lessons Learned

- Students preferred JADE. They found documentation better and API easier to use.
- Both systems had significant learning curves.
- Most (all?) of the time was spent writing software and debugging rather than designing communication protocols.
- This assignment was dropped from the last class taught.

Theory and Algorithms

- Topics covered include
  - notation for describing an agent,
  - agent architectures,
  - game theory,
  - auctions,
  - coordination,
  - voting,
  - learning in multiagent systems.
- Both Weiss and Wooldridge textbooks cover roughly the same material.
- Both fail to provide consistent notation for all aspects of multiagent design (not easy!).
- Vlassis does a better job and includes mechanism design.

The Semantic Web

- Web Services and the Semantic Web are here to stay: RMI, SOAP, WSDL, UDDI, WSDL, BPEL4WS, OWL, OWL-S.
- FIPA has been absorbed by the W3C.
- Part of the standard software engineering curriculum.
- Multiagent aspects are best learned after understanding the technologies as above.
- Many students interested in client/server software engineering problem.
- Not enough time!

The Semantic Web

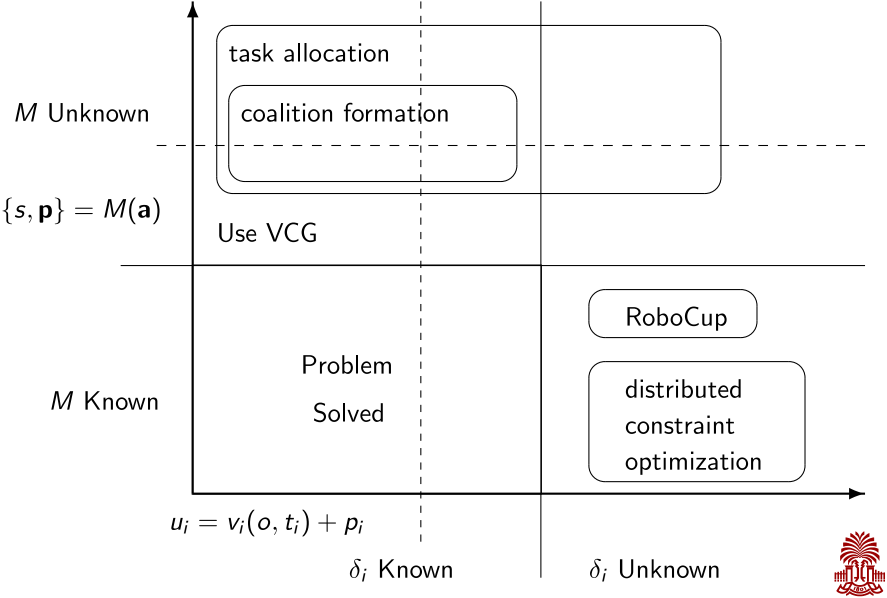
- Web Services and the Semantic Web are here to stay: RMI, SOAP, WSDL, UDDI, WSDL, BPEL4WS, OWL, OWL-S.
- FIPA has been absorbed by the W3C.
- Part of the standard software engineering curriculum.
- Multiagent aspects are best learned after understanding the technologies as above.
- Many students interested in client/server software engineering problem.
- Not enough time!
- Decision: These technologies will be taught as part of a *"Distributed Programming"* class which also covers software agents.

Software Tools

- Will continue to use **RoboCup** and **NetLogo**.
- NetLogo gives hands-on experience with a myriad of algorithms and encourages experimentation—good for understanding algorithms.
- RoboCup provides a much richer environment—good for understanding complexities of real-world systems.

Towards a Unifying Notation of Multiagent Systems

- **Mechanism design** offers us a notation for describing the problem faced by a designer of a multiagent with *selfish* agents.
- It is based on utility theory.
- Can we extend the notation to cover all multiagent systems?



task allocation

coalition formation

M Unknown

$\{s, \mathbf{p}\} = M(\mathbf{a})$

Use VCG

RoboCup

Problem

Solved

M Known

distributed

constraint

optimization

$u_i = v_i(o, t_i) + p_i$

δ_i Known

δ_i Unknown

Conclusion

- Software agents are now a software engineering concern. Web services and the Semantic Web integrate multiagent research. Enough material for a programming class.
- Multiagent research continues to find new (and more complex) algorithms and coordination mechanisms. Tools like NetLogo make it easier for us to understand how they work.
- A unifying notation would help in teaching theory. Mechanism design might be the first step.